(1)

AD

AD-E401 855

TECHNICAL REPORT ARPAD-TR-88005

THE COMPLEXITY ANALYSIS TOOL

PAUL E. JANUSZ

OCTOBER 1988

US ARMY
ARMAMENT MUNITIONS
& CHEMICAL COMMAND
ARMAMENT RESEARCH CENTER

U. S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER
PRODUCT ASSURANCE DIRECTORATE
PICATINNY ARSENAL, NEW JERSEY

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

The citation in this report of the names of commercial firms or commercially available products or services does not constitute official endorsement by or approval of the U.S. Government.

Destroy this report when no longer needed by any method that will prevent disclosure of contents or reconstruction of the document. Do not return to the originator.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER Technical Report ARPAD-TR-88005	5. MONITORING ORGANIZATION REPORT NUMBER
6a. NAME OF PERFORMING ORGANIZATION ARDEC, PAD 6b. OFFICE SYMBOL AMSMC-QAH-A(D)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Materials and Technology Laboratory
6c. ADDRESS (CITY, STATE, AND ZIP CODE) SQA/Math Branch Picatinny Arsenal, NJ 07806-5000	7b. ADDRESS (CITY, STATE, AND ZIP CODE) SLCMT-MSI-QA, Mr. F. Stenton Watertown, MA 02172-0001
8a. NAME OF FUNDING/SPONSORING ORGANIZATION ARDEC, IMD STINFO Br 8b. OFFICE SYMBOL SMCAR-IMI-I	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (CITY, STATE, AND ZIP CODE) Picatinny Arsenal, NJ 07806-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 78011A PROJECT NO. DE51 TASK NO. 3094 WORK UNIT ACCESSION NO.
11. TITLE (INCLUDE SECURITY CLASSIFICATION) THE COMPLEXITY ANALYSIS TOOL	
12. PERSONAL AUTHOR(S) Paul E. Janusz	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM 1984 TO 1987	14. DATE OF REPORT (YEAR, MONTH, DAY) October 1988 15. PAGE COUNT 26

16. SUPPLEMENTARY NOTATION
This project was accomplished as part of the U.S. Army's Manufacturing Methods and Technology Program. The primary objective of this program is to develop, on a timely basis, manufacturing processes, techniques, and equipment for use in production of Army material.

17. COSATI CODES			18. SUBJECT TERMS (CONTINUE ON REVERSE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)		
FIELD	GROUP	SUB-GROUP	Software complexity	Structured testing	Software verification & validation
			Test paths	Unit level testing	Automated tool
			MMT - Manufacturing methods and technology		

19. ABSTRACT (CONTINUE ON REVERSE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

This report presents an overview of the complexity analysis tool (CAT), an automated tool which will analyze mission critical computer resources (MCCR) software. CAT is based on the cyclomatic complexity metric, which is used to measure, quantify, or evaluate a software module's complexity. Software which is less complex is easier to maintain and is less likely to have embedded errors. The metric suggests the minimum number of paths which must be tested in order to assure software reliability. The ideal limit of complexity is 10 for any software module. A module of complexity greater than 10 would need to be modified or redesigned.

Applied during software development, the complexity measure limits the number of independent paths in a program at the design and coding stages so that testing will be manageable during the later stages. This allows for structured testing, avoiding the problems arising from software which is inherently untestable. This testing technique can be applied during all stages of testing (i.e., unit, integration, and qualification testing). (cont)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL I. HAZNEDARI	22b. TELEPHONE (INCLUDE AREA CODE) (201) 724-3316 22c. OFFICE SYMBOL SMCAR-IMI-I

DD FORM 1473, 84 MAR

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT: (cont)

CAT automates the metric for BASIC (HP-71), ATLAS (EQUATE), Ada (subset of DOD-STD-1815A), Ada PDL, and PDL (Caine, Farber, Gordon). It operates on both an IBM PC-AT (running MS DOS) and on DEC VAXs (750/780 running AT&T UNIX 5.2). CAT analyzes source code and computes complexity on a module basis. CAT also generates graphic representations of the logic flow paths and test paths, as well as other textual output.

CONTENTS

FIGURES

INTRODUCTION

Approximately 85% of today's weapon systems employ embedded computers, and there is a trend towards more complex systems. In these mission critical computer resources (MCCR), a software error can have a drastic effect upon system performance.

There are several problems facing MCCR development. First, there is the ever increasing hardware dependency upon the software to successfully carry out its mission. At the same time, software development is a relatively new and unproven technology compared to hardware. Software development is often performed in a sloppy fashion and is improperly documented. Use of off-the-self software is often force fitted to the application at hand. This approach to software design leads to errors, and if detected late in the development cycle can drive up costs substantially.

The major problem with DoD software development projects today relates to the verification and validation of requirements in the software program. Statistics have shown that approximately 46% to 64% of software errors are traced back to inadequate requirements and design. Of these errors, 70% of them are not caught early in the life cycle and propagate into production and deployment (ref 1). The problem stems from the fact that many software implementations of functional requirements remain untested, and consequently errors are found during use when an untested path in the software is executed. The surprises will cost the government both time and money when extensive debugging and reverification efforts are required to fix these problems. The cost of correcting these errors can be as much as 300 times the cost to correct it during unit testing.

Software assessments for the most part are subjective in nature. The difference between hardware and software quality assessments is the lack of measurable parameters for software. For this reason there is an increasing drive to develop and apply techniques which provide a quantitative means of measuring or assessing software quality.

The primary means of performing software quality assessments is through independent verification and validation (IV&V). This includes verifying that requirements are met, through qualitative specification reviews, having adequate documentation, and testing. Testing in support of software quality assurance (SQA) is very labor intensive. In fact, the norm for software testing is about 50% of the total software development effort (ref 1). In order to meet program deadlines and cost constraints, the testing effort is often cut short, leaving doubt as to the quality of the software.

The problems mentioned above were addressed by the Software Quality Assurance/Math Branch of the U.S. Army Armament Munition and Chemical Command (AMCCOM). A technique was devised that would help verify the quality of the software through the use of quantitative measures. The assessment technique in the form of an automated tool would increase productivity and efficiency of available manpower, reduce subjectivity, reduce fielded system failures, and consequently would reduce development and maintenance costs. The result of this effort was the automation of the cyclomatic complexity metric.

THE CYCLOMATIC COMPLEXITY METRIC

The cyclomatic complexity metric is documented in the U.S. Department of Commerce National Bureau of Standards (NBS) Special Publication 500-99, "Software Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric" (ref 2). It is based upon structured programming conventions and graph theory. The idea behind the metric is to measure, quantify or evaluate the complexity of a software module. Software which is less complex can be comprehended, is easier to maintain, can be tested thoroughly, and is less likely to have embedded errors. The ideal limit of complexity is 10 for any software module. A module of complexity greater than 10 would need to be broken down into smaller submodules.

The concept behind the metric is simple; one counts the number of control tokens which exist in the software module to determine complexity. Complexity can be calculated as:

Complexity = The number of control tokens + 1

Control tokens are programming language statements which in some way provide provision points which modify the top-down flow of the program. In other words, statements such as IF-THEN-ELSE, CASE, GOTOs, are considered to be control tokens since they base program flow upon a logical decision, thereby creating alternate paths which program execution may follow. Thus, at the same time, the technique also identifies the critical paths needed to exercise every line of code in the module. A module of complexity five would have five critical or basis paths. These paths can then be used to adequately test software modules while minimizing the extent of testing required.

The metric can be used throughout the entire software life cycle. Applying a limited complexity as a contractual requirement will force structured programming techniques. Applied during software development, the metric will limit the number of basis paths in a

program at the design and coding stages. It can be used during software testing to identify the basis paths and to minimize the testing effort. During the maintenance phase, a proposed change should not be allowed to substantially drive up the complexity, whereby increasing the testing effort.

The cyclomatic complexity metric allows you to quantitatively assess the software. As discussed earlier, having this metric incorporated in the form of an automated tool would have substantial benefits as well.

COMPLEXITY ANALYSIS TOOL

The complexity analysis tool (CAT) is an automated tool which is based upon the cyclomatic complexity metric. It is designed to run on an IBM PC AT under a MS DOS operating system, as well as on a DEC VAX under a UNIX (AT&T 5.2) operating system. CAT will analyze an ASCII source code file and will identify the various control tokens. It will then generate a graphic representation of the logic flow paths, called a data flow diagram (DFD), similar to a flow chart. The basis paths can also be displayed. CAT's interface consists of a series of user-friendly menus which guide the user through execution of the tool.

The first thing CAT will ask for is the language being analyzed. CAT currently has the ability to analyze programs written in BASIC (HP-71), PDL (Caine, Farber, Gordon), Equate ATLAS, Ada (DOD-STD-1815A), and Ada PDL. After selecting an appropriate file, the tool will pass the file through the appropriate language parser and perform the metrics analysis. CAT operates under the assumption that the file can be compiled successfully. If not, an appropriate error message will be raised. Upon successful completion of the parser and metrics analysis, CAT is ready to display its output. The user, through the use of menus, selects how the information is to be displayed (i.e., to the screen printer, plotter, or disk file). Examples are shown in figures 1 through 5.

CAT's output consists of several tables and diagrams, including the source listing, data flow diagram, and test paths. This output will provide the developer or assessor a pictorial and quantitative representation of the software logic. The first output consists of a source listing of the entire file. This listing contains two major sections. The first is the module directory (fig. 1). It lists the modules found in this file and presents the vital statistics for each of the modules. Modules are listed in the order they were found in the source file, i.e., by ascending line number. The directory shows the name of each module, the starting line, the number of lines in the module, and cyclomatic complexity. At the far left of each line in the directory, a letter is given to the module. This letter is used in the body of the listing to identify code belonging to the module.

The second section is the listing itself (fig. 2). The leftmost columns of each line are used to show the correspondence between the source code and the DFD. First is the line number, which is the count of lines from the top of the file. Next is the module letter which identifies the module containing this line of code. Following the module letter is the list of nodes that are represented, wholly or partially, by this line. Each module may be examined individually if desired.

In looking at the DFD (fig. 3), there is a summary of information at the top of the DFD. Listed are the source filename, the module within the source file being analyzed, the module's complexity, the number of lines of code in that module, and the date and time of the analysis. Also included is a color scheme for the DFD to indicate program flow direction.

The numbers on the DFD represent nodes, a block of statements where the program flow is sequential. Edges represent the program's branches taken between blocks. Edges that cause loops are shown in one color. These edges always flow from the bottom to the top of the page. Edges that perform a structured exit of a loop (such as a WHILE or FOR statement) are shown in another color. These edges flow from the top to the bottom of the page. All remaining edges are drawn in a third color. These edges also go from the top to the bottom of the page. Another indication of the direction and function of an edge is its shape. Loops or loop exits are always drawn as curved lines. Other edges are drawn as straight lines unless they must be curved to avoid colliding with another node.

If a file analyzed contained several modules, and one of these modules has a complexity greater than 10, or one of these modules has been changed, these modules would be candidates for further review. By going through the menu interface, a specific module can be selected for individual review. The module's corresponding source listing, DFD, and basis paths can be examined. CAT can automatically determine the basis paths and display them in two fashions. The first is by a series of numbered nodes, such as 0-1-2-3-5-6-7-14-7-8-9-10-13-3-4-16-17, corresponding to the DFD (fig. 4). The second means is by graphing the test paths individually (fig. 5).

BENEFITS

The significant benefits derived from a quality design are realized throughout the full life-cycle of the program (i.e., development, production, post-deployment), as opposed to benefits derived from an instantaneous assessment. CAT provides an overall improvement in design by enforcing structured programming techniques upon the software programmer, thus designing quality into the software. Imposition of the metric would also allow early inspection and diagnosis of the problem areas in the software logic. For example, during initial design, CAT can be used to assure a low complexity in the PDL. By using the PDL, DFDs, source listing and test paths, one can perform a

walk-through to check for logic or function errors before coding takes place. When identified problems are resolved, proceed to the coding phase. The developed code can then be passed through CAT again, coming up with another set of output. These two sets of output, one from the PDL and one from the code, can then be compared against one another. There should not be significant differences between the two. For example, if a PDL module had a complexity of five, the code implementing that module should not have a corresponding complexity of 25. The complexities are likely to increase implementing the PDL into code, but they should not be significant differences. If there are differences such as these, you know that the requirements in the PDL are not correctly implemented into the code. This is another way errors could be uncovered early in the life cycle before they propagate to unmanageable portions.

A major benefit derived from the use of the metric would be an improvement in the ease and efficiency of testing. By concentrating on the basis paths, the testing effort is prioritized and minimized. The test paths as a whole can be used for unit testing of the module and can be part of its unit development folder. The tool can also be used in conjunction with acceptance testing as a means of verifying the performance of the software.

From a post deployment perspective, the metric is a means of obtaining a measure of software supportability. Suppose you would like to know the effects of a proposed change in the software. In looking at figure 2 you could locate the lines of source code you intend to change. By looking at the corresponding node letters on the left-hand side, you could then go to figure 3 and note its corresponding effect upon the other nodes. If the area in question is highly structured, a change would probably have little impact. However, if the area was highly unstructured, a change would probably have a drastic impact. The diagrams could also be used in a reverse fashion. For example, if you notice that a particular area of the DFD was cluttered and you wanted to clean it up, you could note which nodes were involved. You could then go to figure 2 and find the corresponding source code you would have to change.

CAT, because of its data flow analysis, not only detects a modification but relates the modification to a particular flow area in the program. To fully characterize a program change, CAT can flag changes in the program flow paths. Thus the complexity metric will provide an efficient means of identifying regression test cases to verify those portions of the software program which are changed.

CAT as an automated tool can eliminate manual computation errors, eliminate subjectivity, and significantly reduce the number of manhours required for manual computations of the metric. Preliminary forecasts estimate that the effort required to apply the metric manually to a software development process is approximately 3% to 7% of the overall development effort (assuming no learning curve is required). This estimate is based on our own in-house experience with MCCR software using manual calculations of complexity. We project that with the automated tool the effort would drop

by about one or two orders of magnitude. In either case, taking the extra effort to use the metric provides a vehicle for detecting and correcting bugs earlier in the life cycle. This in turn could amount to significant benefits and savings in terms of development and testing costs, productivity gains, and software quality and reliability.

CONCLUSIONS

The use of CAT as a quality assurance tool provides a quantitative measure of software quality, structure, robustness, testability, and maintainability. Imposition of the cyclomatic complexity metric on the developer during the early phases of the life cycle will result in a quality design. Benefits will then be realized throughout the life cycle of the program.

The complexity metric has been incorporated as a requirement in several mission critical computer resources programs currently underway. A follow-on study will be performed which will quantify these benefits in terms of time, labor, and cost savings, as well as correlations between complexity and reliability, error rate, modifiability, etc. The study will be performed using data gathered from users of the metric and tool. The complexity analysis tool can be obtained by contacting:

U.S. Army AMCCOM
AMSMC-QAH-A(D)
Bldg 62
Picatinny Arsenal, NJ 07806-5000
(201) 724-4849, Autovon 880-4849

Complexity Analysis Tool
Listing from source file xsample.ada

Module letter	Module name	Cyclomatic complexity	Starting line	Number of lines
A	SAMPLE_ADA	6	6	19

Figure 1. Module directory

Page 1 CAT listing Source file:xsample.ada

Line	Module/Node	Source Text

```
Page 1        CAT Listing          Source file:xsample.ada
Line  Module Node                Source Text

1                                procedure SAMPLE_ADA is
2                                     FIRST: INTEGER;
3                                     NEXT: INTEGER;
4                                     C: CHARACTER;
5
6       A0                       begin
7       A1                          if FIRST=0 then
8       A15                            put line(" First equal to zero ");
9       A2                          else
10      A2                             NEXT:= FIRST;
11      A3                                while (NEXT /=0) loop
12      A5    A6                             put_line(" Next not equal to zero ");
13      A7                                   for I in 1..20 loop
14      A14                                     C:=C + 1;
15      A14                                     put(C);
16      A8    A14                            end loop;
17      A9                                   case C is
18      A12                                     when 1 => ADD;
19      A11                                     when 2 => DELETE;
20      A10                                     when others => PRINT OUT;
21      A13                                end case;
22      A4    A13                       end loop;
23      A16                          end if;
24      A17                       end SAMPLE_ADA;
```

Figure 2. Module listing

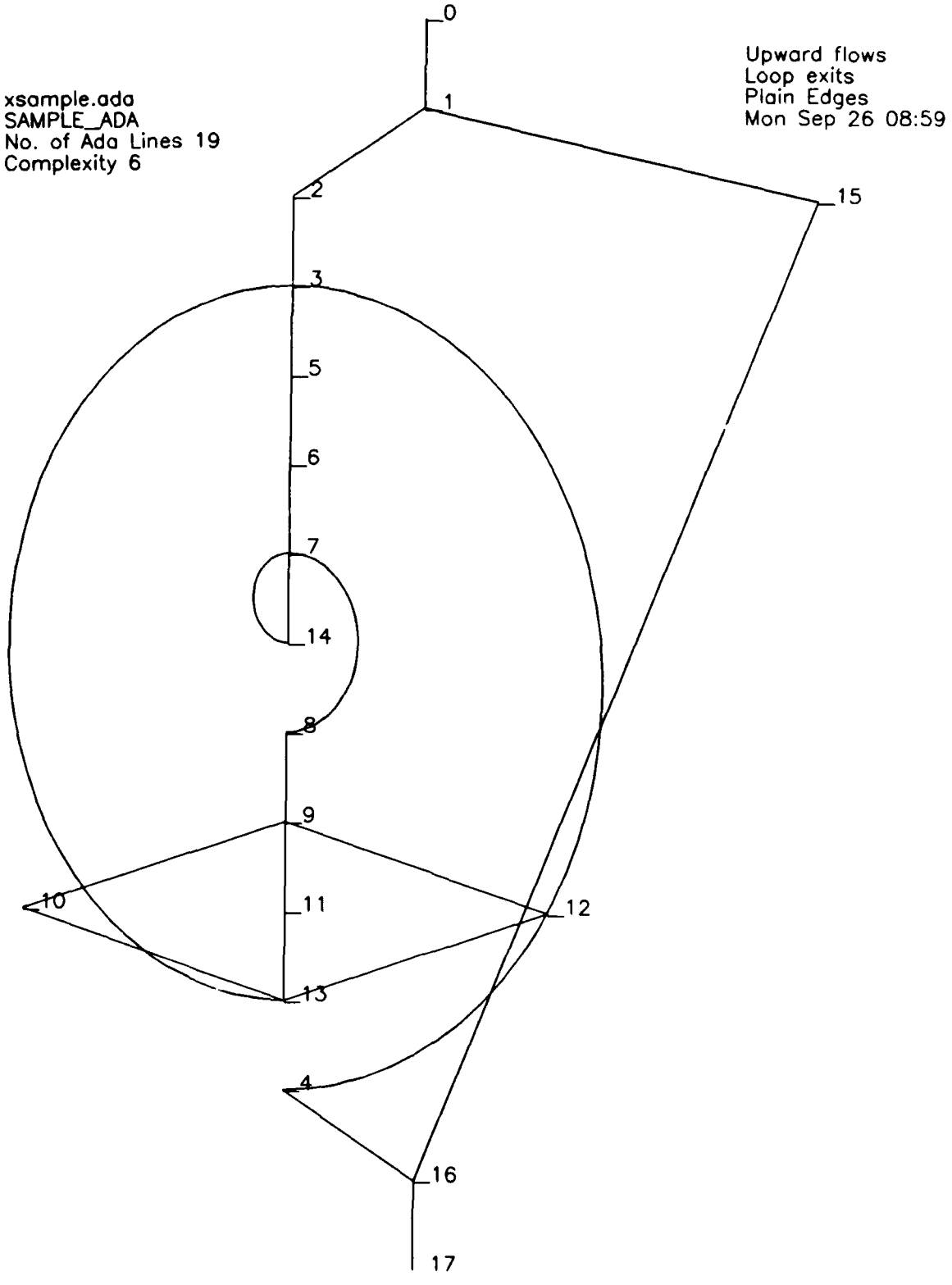

Figure 3. Data flow diagram

Test Path Listing

Module Name: SAMPLE_ADA
Complexity 6
Language: Ada

File: xsample.ada
Date/Time: Mon Sep 26 09:38
Page 1

Baseline: 0 1 15 16 17

Test Path 1: 0 1 2 3 4 16 17

Test Path 2: 0 1 2 3 5 6 7 8 9 10 13 3 4 16 17

Test Path 3: 0 1 2 3 5 6 7 14 7 8 9 10 13 3 4 16 17

Test Path 4: 0 1 2 3 5 6 7 8 9 12 13 3 4 16 17

Test Path 5: 0 1 2 3 5 6 7 8 9 11 13 3 4 16 17

Figure 4. Test path listing

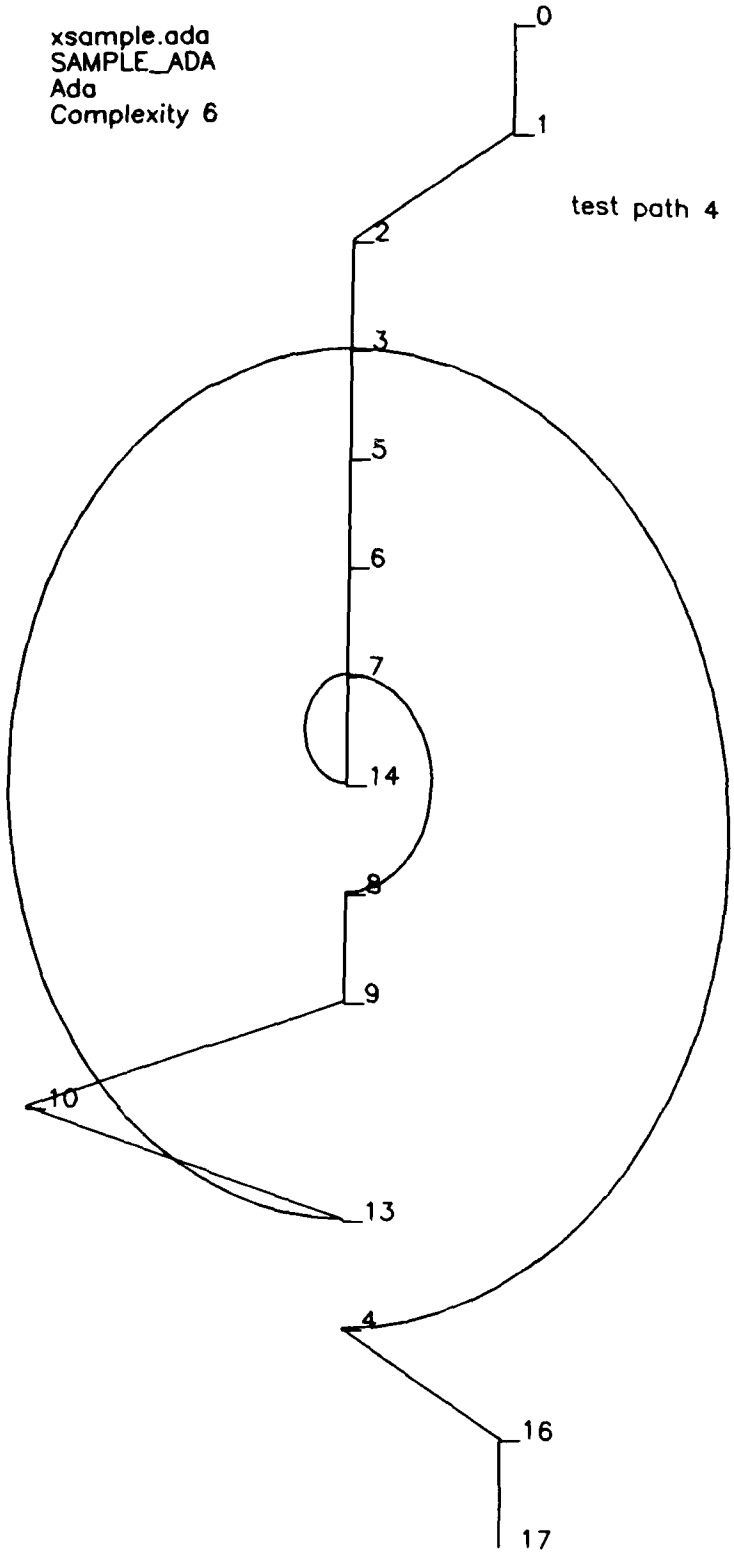

Figure 5. Test path graph

REFERENCES

1. Sorkowitz, Alfred, Software Quality Assurance and Testing, presented at the Testing Computer Software Conference, Washington, D.C., October 1984.

2. U.S. Department of Commerce, National Bureau of Standards, "Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric," NBS Special Publication 500-99, December 1982.

BIBLIOGRAPHY

1. McCabe, T. J. (ed.), "Structured Testing," IEEE Computer Society Press, IEEE Catalog No. EH0200-6, 1982.

2. Walsh, T. J., "A Software Reliability Study Using a Complexity Measure," Proceedings of the 1979 National Computer Conference, AFIPS Press, 1979.

3. Janusz, Paul E. and Turoczy, William R., "Application of Software Test Tools to Battlefield Automated Systems, Phase I," Technical Report ARPAD-TR-84003, ARDEC, Dover, NJ, 1984.

DISTRIBUTION LIST

Commander
Armament Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCAR-IMI-I (5)
Picatinny Arsenal, NJ 07806-5000

Commander
U.S. Army Armament, Munitions and Chemical Command
ATTN: AMSMC-GCL(D)
AMSMC-PBM-TP(D), Tom McWilliams
AMSMC,FSC(D), Dan Nathan
Dave Johnson
AMSMC-FSD, William Ginley
AMSMC-QAA(D), G. DeMassi
AMSMC-QAH-A(D), Magid Athnasios
Michael Bucknor
David Castellano
Michael Cernek
Joseph Gombos
Mark Herbst
Paul Janusz
Wayne Lee
Patricia Lyon
Sharyn DcDowell
Aron Dutta
Geza Pap
Elizabeth Parliman
Richard Payne
Albert Stanbury
Allison Willis
Paul Willson
AMSMC-QAH-T(D), V. Minetti
G. Edick
AMSMC-QAR-I(D), M.H. Weinberg
AMSMC-SCM-P(D), J. Beetle
Picatinny Arsenal, NJ 07806-5000

Commander
Army Armament Research, Development and Engineering Center
ATTN: AMCPM-CAWS(D)
AMCPM-AL(D)
AMCPM-MCD(D)
AMCPM-TMA(D)
AMCPM-MO(D)
AMCPM-FZ(D)
AMCPM-HIP(D)
Picatinny Arsenal, NJ 07806-5000

Administrator
Defense Technical Information Center
ATTN: Accessions Division
Cameron Station
Alexandria, VA 22304-6145

Director
U.S. Army Materiel Systems Analysis Activity
ATTN: AMXSY-MP
AMXSY-R, William Clay
AMXSY-RM, John Woodworth
Aberdeen Proving Ground, MD 21005-5066

Commander
Chemical Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCCR-MSI
Aberdeen Proving Ground, MD 21010-5423

Commander
Chemical Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Commant
ATTN: SMCCR-RSP-A
Aberdeen Proving Ground, MD 21010-5423

Director
Ballistic Research Laboratory
ATTN: AMXBR-OD-ST
Aberdeen Proving Ground, MD 21005-5066

Chief
Benet Weapons Laboratory, CCAC
Armament Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCAR-CCB-TL
Watervliet, NY 12189-5000

Commander
U.S. Armament, Munitions and Chemical Command
ATTN: SMCAR-ESP-L
Rock Island, IL 61299-6000

Director
U.S. Army TRADOC Systems Analysis Activity
ATTN: ATAA-SL
White Sands Missile Range, NM 88002

Commander
U.S. Army Materiel Command
ATTN: AMCPD-IP, T. Shifflet
W.J. Jenkins
AMCPD-P, D. Griffin
AMCQA, S. Lorber
J. Stahl
AMCQA-E, Chris Neubert
AMCQA-EQ, E. Soliven
AMCQA-P, E. Lesser
5001 Eisenhower Avenue
Alexandria, VA 22333-0001

Commander
U.S. Army Armament, Munitions and Chemical Command
ATTN: AMSMC-QA(R), L. Griffin
AMSMC-QAK-B(R), R. Fer
Rock Island, IL 61299-6000

Director
U.S. Army Industrial Base Engineering Activity
ATTN: AMXIB-PA, D. Brim
AMXIB-PS, G. Fisher
L. Gross
Rock Island, IL 61299-7250

U.S. Army Belvoir R&D Center
ATTN: STRBE-V, B. Wells
Fort Belvoir, VA 22060-5606

Commander
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCPB-QAL, V. Warren
Pine Bluff Arsenal
Pine Bluff, AR 71602-9500

Commander
U.S. Army Aviation System Command
ATTN: AMSAV-QE, R. R. L'Italian
Letterkenny Army Depot
Chambersburg, PA 17201-4170

Commander
U.S. Army Communications Electronics Command
ATTN: AMSEL-PA, A. D'Angelo
AMSEL-PA-MT-S, P. Kogut
Fort Monmouth, NJ 07703-5023

Commander
U.S. Army Laboratory Command
ATTN: AMSLC-AS, S. Alster
AMSLC-CT, R. Moore
2800 Powder Mill Road
Adelphi, MD 20783-1197

Commander
U.S. Army Missile Command
ATTN: AMSMI-Q, T. Howard III
AMSMI-QET, T. McVey
ASMI-RD-SE, L. Ross
L. Daniels
AMSMI-RKC, B. J. Alley
AMSMI-RKP, J. Wright
Redstone Arsenal, AL 35898-5720

Commander
U.S. Army Center for Night Vision and Electro-Optics
ATTN: AMSEL-PA-EN, V. Burger
A. Vuille
AMSEL-RD-NV-TS, R. Stefanik
Fort Belvoir, VA 22060-5606

Commander
U.S. Army Test and Evaluation Command
ATTN: AMSTE-TC-M, K. Balliet
J. Piro
R. G. Shelton
Aberdeen Proving Ground, MD 21005

Commander
U.S. Army Armament, Munitions and Chemical Command
ATTN: AMSMC-QAC-E(E), H. Elbaum
W. Maurits
Aberdeen Proving Ground, MD 21010-5423

Commander
Watervliet Arsenal
ATTN: SMCWV-PPI, W. Garber
SMCWV-QAE, S. Krupski
SMCWV-QA, J. Miller
Watervliet, NY 12189-5000

Commander
U.S. Army Natick R&D Center
ATTN: STRNC-R, R. Day
Kansas Street
Natick, MA 01760-5014

Commander
U.S. Army Tank-Automotive Command
ATTN: AMSTA-Q, L. Barnett
AMSTA-QAT, F. Braun
Robert Crow
D. Gamache
AMSTA-RCK, J. Chevalier
AMSTA-RCKM, D. Ostberg
Warren, MI 48397-5000

Commander
U.S. Army Troop Support and Aviation
Material Readiness Command
ATTN: AMSTS-Q, W. G. Creel
4300 Goodfellow Boulevard
St. Louis, MO 63120-5720

U.S. Army TROSCOM
ATTN: AMSTS-Q
St. Louis, MO 63120

U.S. Army TROSCOM-BRDEC
ATTN: STRBE-TQ, Lawrence Makowsky
Ft. Belvoir, VA 22060-5606

U.S. Army TECOM-WSMR
ATTN: STEWS-TE-OE, Marthe Wygant
White Sands, NM 88002-5070

U.S. Army ALMC
ATTN: AMXMC-ACM
Ft. Lee, VA 23801

U.S. Army DLA
ATTN: DLA-QES, Armond Darrin
Cameron Station
Alexandria, VA 22304-6100

U.S. Army DCASR
ATTN: DCASR-PHI-QTX, Ann Quinn
2800 S. 2oth St.
Philadelphia, PA 19101

U.S. Army DCASR-NY
ATTN: DCASR-QT, Mike Spezzaferro
201 Varick St.
New York, NY 10014

U.S. Army SE&L
ATTN: AMXMC-SEL-E, David Jenkins
Texarkana, TX 75507-5000

U.S. AMETA
ATTN: AMXOM-QA, Ray Loecke
Bruce Brocka
Rock Island, IL 61299-7040

USASDC
ATTN: DASD-H-TP, Emmett Magathan
Patrick Duggan
P.O. Box 1500
Huntsville, AL 35807-3801

USA SIGCEN
Maj Mourfield
Chief ACSD
ATTN: ATZH-CDC
Ft. Gordon, GA 30905

Director
Materials Technology Laboratory
ATTN: SLCMT-D
SLCMT-DD
SLCMT-MS
SLCMT-OM
SLCMT-MC
SLCMT-TP
SLCMT-MSI
SLCMT-MSI-NE
SLCMT-OMP
SLCMT-MCM-SB
Watertown, MA 02172-0001

Commander
U.S. Army Dugway Proving Ground
ATTN: STEDP-MT-AT, K. Dumbauld
STEDP-MT-C-T, F. Bagley
STEDP-PO, J. McKenzie
Dugway, UT 84002-5000

Commander
Harry Diamond Laboratories
ATTN: SLCHD-PO-P, J. Hoke
2800 Powder Mill Road
Adelphi, MD 20783-1197

Commander
U.S. Army Electronic Technology and Devices Laboratory
ATTN: SLCET-R, J. Key
Fort Monmouth, NJ 07703-5302

Commander
U.S. Army Yuma Proving Ground
ATTN: STEYP-MTD, W. E. Brooks
Yuma, AZ 85365

Commander
U.S. Army White Sands Missile Range
ATTN: STEWS-QA, C. Treat
White Sands Missile Range, NM 88002

Commander
U.S. Army Tropic Test Center
ATTN: STETC-LD-M
APO
Miami, FL 34004

Commander
Anniston Army Depot
ATTN: SDSAN-DQA, Mr. Pennington
Anniston, AL 36201

Commander
U.S. Army Depot System Command
ATTN: AMSDS-QM, B. Newman
Chambersburg, PA 17201-4170

Commander
U.S. Army Pine Bluff Arsenal
ATTN: AMCPB-QA, H. Love
Pine Bluff, AR 71611

Commander
Corpus Christi Army Depot
ATTN SDSCC-Q, D. L. Ross
Corpus Christi, TX 78419

Commander
U.S. Army Jefferson Proving Ground
ATTN: STEJP-TD, MAJ A. Alqhin
Madison, IN 47250

Commander
Letterkenny Army Depot
ATTN: SDSLE-Q, G. Mantooth
Chambersburg, PA 17201-4150

Commander
New Cumberland Army Depot
ATTN: SDSNC-Q, A. T. Holderbach
New Cumberland, PA 17070

Commander
Red River Army Depot
ATTN: SDSRR-Q, W. D. Wuertz
Texarkana, TX 75501

Commander
U.S. Army Electronic Proving Ground
ATTN: STEEP-MT, LTC J.R. Sutherland, Jr.
Fort Huachuca, AZ 85613

CommanderLexington-Blue Grass Army Depot
ATTN: SDSLB-QA, J. Palmer, Jr.
Lexington, KY 40511-5100

Commander
Sacramento Army Depot
ATTN: SDSSA-Q, R. Bragg
Sacramento, CA 95813-5027

Commander
Savanna Army Depot Activity
ATTN: SDSLE-VS
Savanna, IL 61074-9636

Commander
Seneca Army Depot
ATTN: SDSSE-R, P.W. Chavez
Romulus, NY 14541

Commander
Sharpe Army Depot
ATTN: SDSSH-Q, J. E. Seyfried
Lathrop, CA 95331

Commander
Sierra Army Depot
ATTN: SDSSI-QA, V. Steed
Herlong, CA 96113

Commander
Tooele Army Depot
ATTN: SDSTE-QA, R.M. Rich
Tooele, UT 84074-5010

Commander
Pueblo Depot Activity
ATTN: SDSTE-PUQ, J. Farley
Pueblo, CO 81001

Commander
Tobyhanna Army Depot
ATTN: SDSTO-Q, W.J. Lord
Tobyhanna, PA 18466

Sandia National Laboratories
ATTN: SQAE, Darl Patrick
Quality Assurance Division 7252
Albequerque, NM 87185

U.S. Army AMCCOM
ATTN: AMSMC-QAV-A, Bill Thetford
Edggewood, MD 21010-5423

U.S. Army AVSCOM
ATTN: AMSAV-Q, Henry Fong
AMSAV-QP, Mark Vail
St. Louis, MO 63120

U.S. Army CECOM
ATTN: AMSEL-PA-DL, Brian Casey
AMSEL-PA-MT-S, Joel Heidelberg
Ft. Monmouth, NJ 07703-5023

U.S. Army DESCOM
ATTN: AMSDS-Q-E-T, Douglas Hanna
Chambersburg, PA 17201

U.S. Army LABCOM
ATTN: AMSLC-EN-PA, John Goon
Adelphi, MD 20783